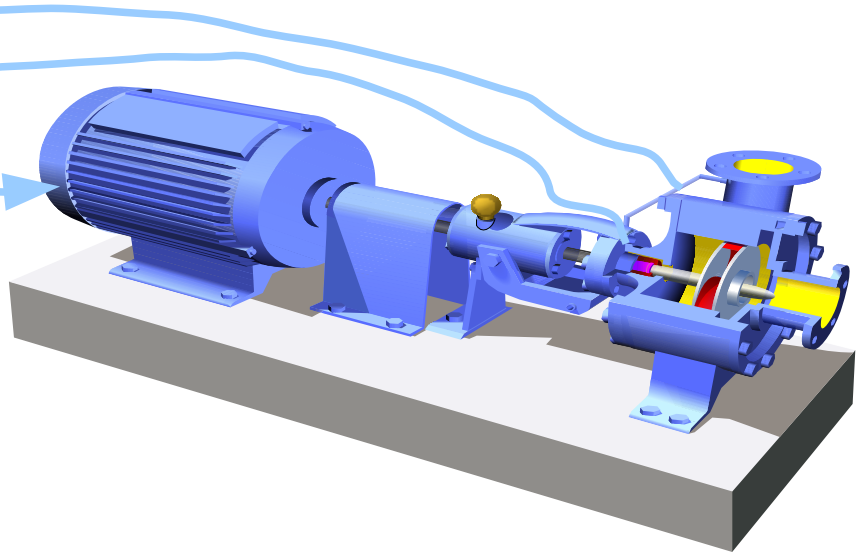
Practical Real-Time Linux

Xenomai and PREEMPT_RT

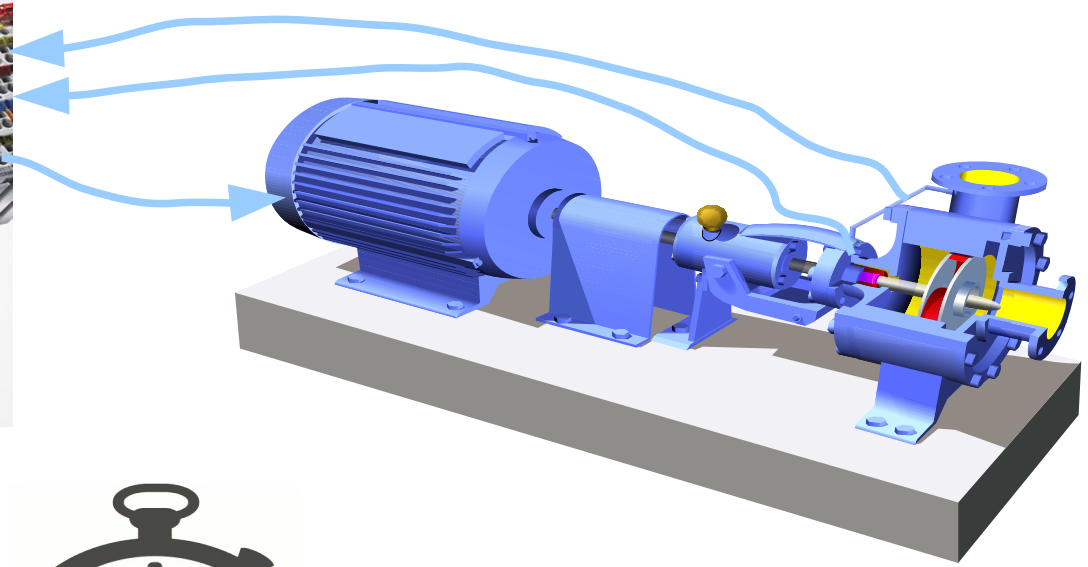
Arnout Vandecappelle **arnout@mind.be**

http://www.mind.be/content/Presentation_Real-Time_Linux.odp

Adding control to a high-pressure pump



Adding control to a high-pressure pump



50µs

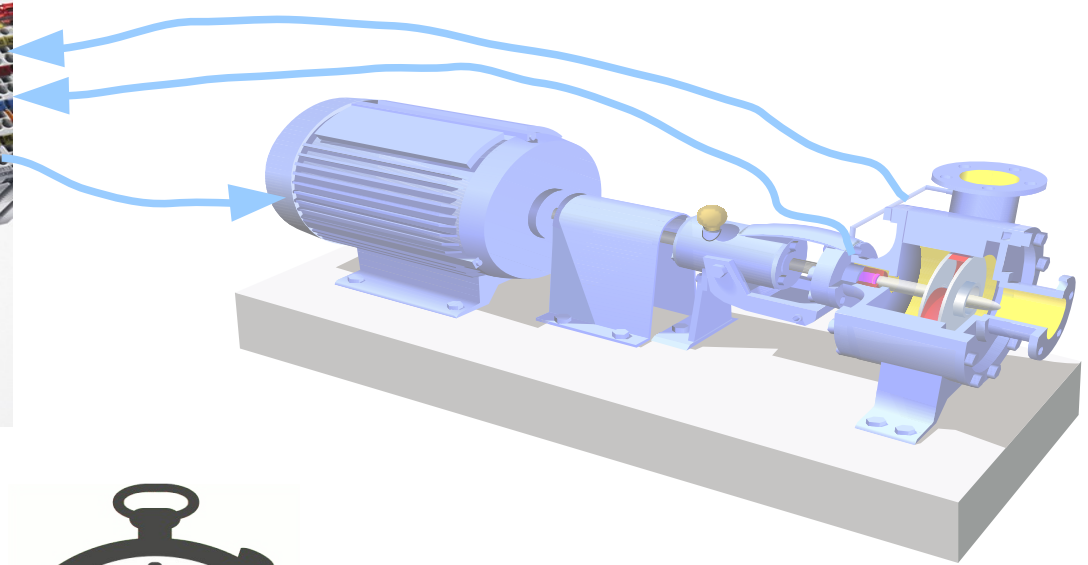
Practical Real-Time Linux

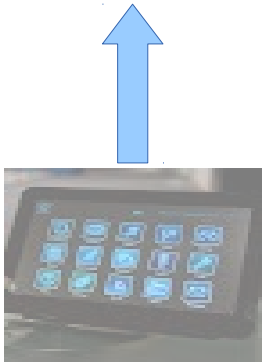
- Xenomai: separate RT and Linux
  - Motor control
- PREEMPT_RT: native RT in Linux
  - GNSS receiver
- Conclusions and future directions

Xenomai

Separate Real-Time and Linux

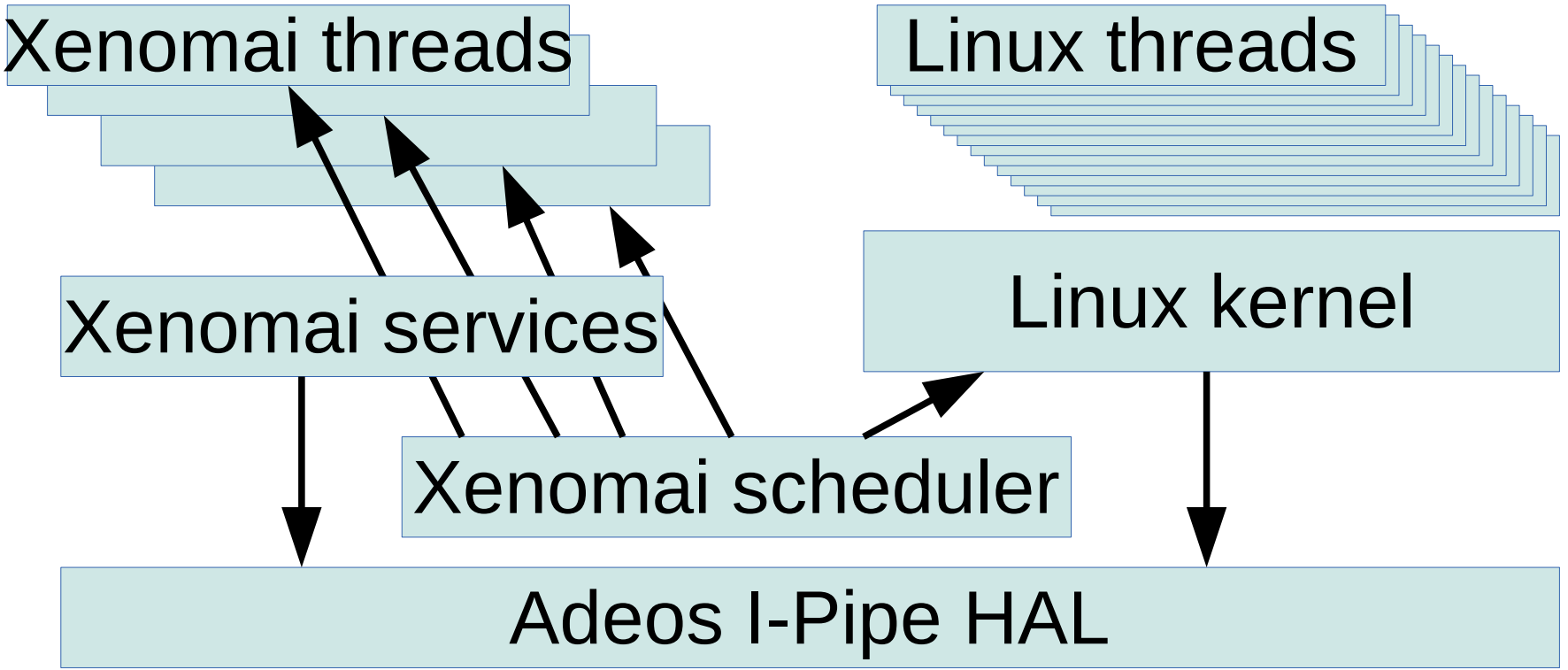
Xenomai separates real-time from Linux scheduler



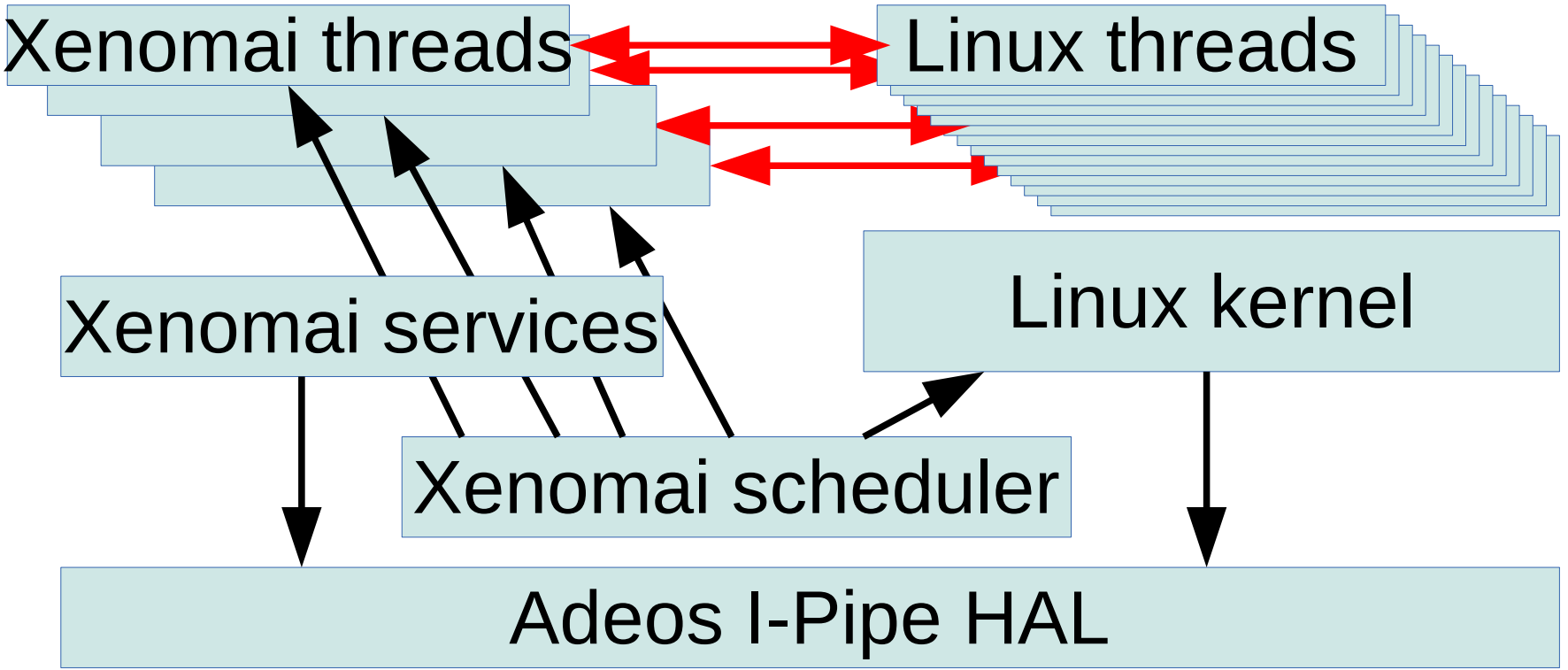


50µs

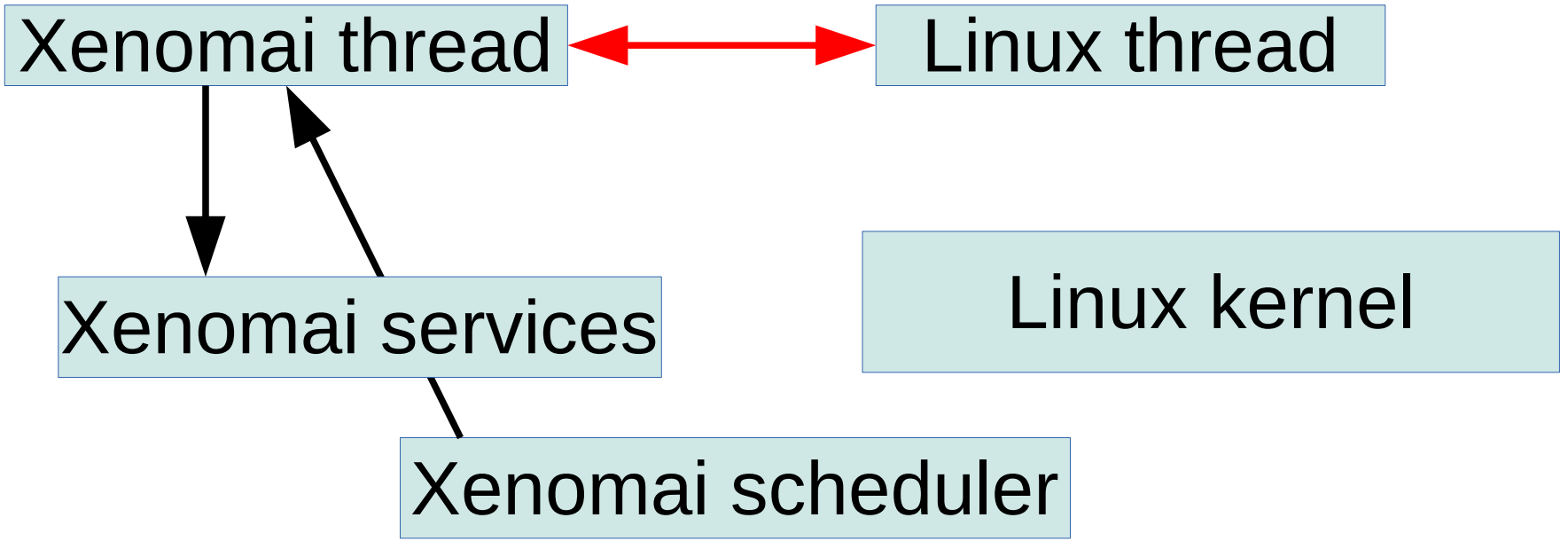
Xenomai has a separate low-latency scheduler



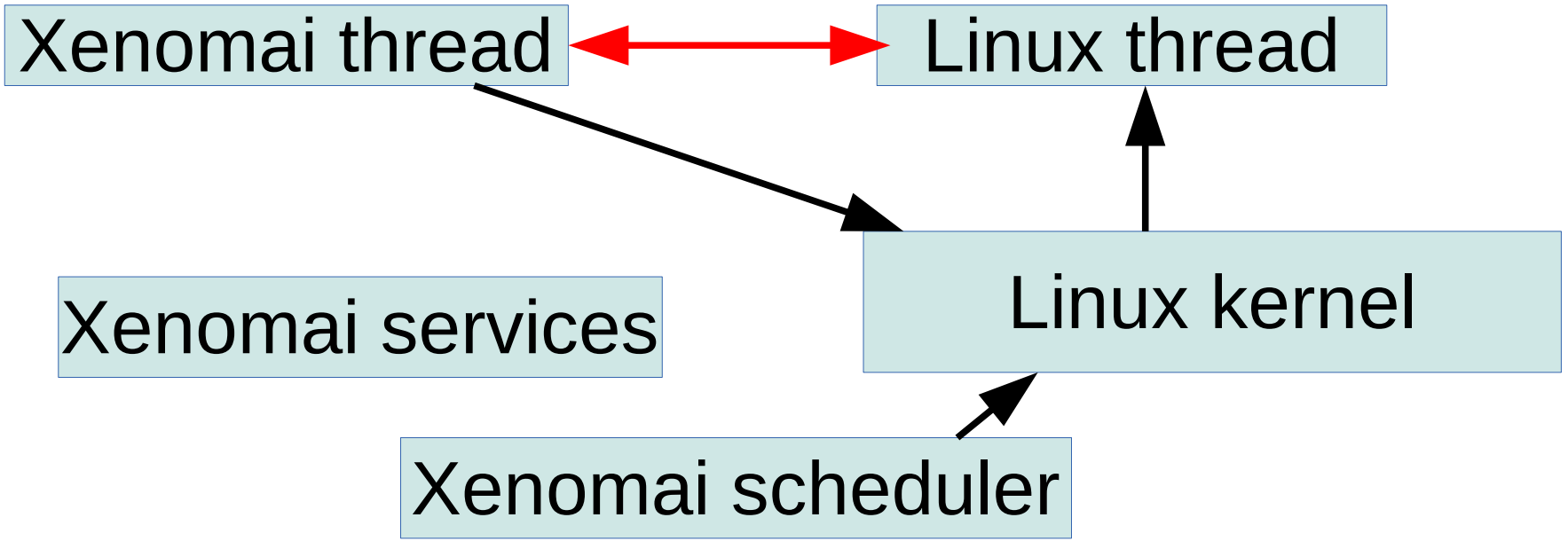
Xenomai threads
shadow Linux threads



Xenomai thread switches to *secondary mode* on Linux syscall



Xenomai thread switches to *secondary mode* on Linux syscall



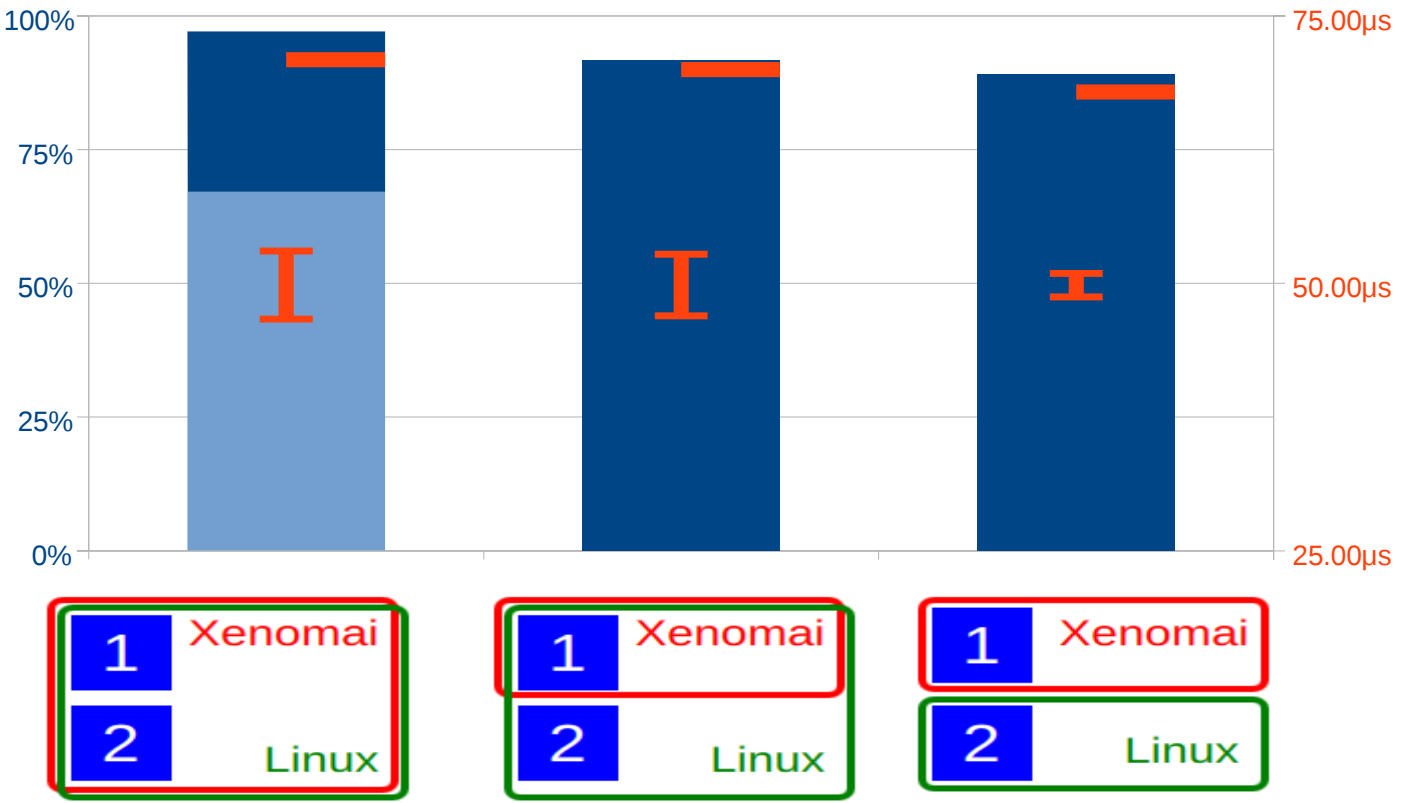
Avoid mode switches to guarantee RT

- Xenomai does a lot out of the box by wrapping POSIX calls
- Check `/proc/xenomai/stat`

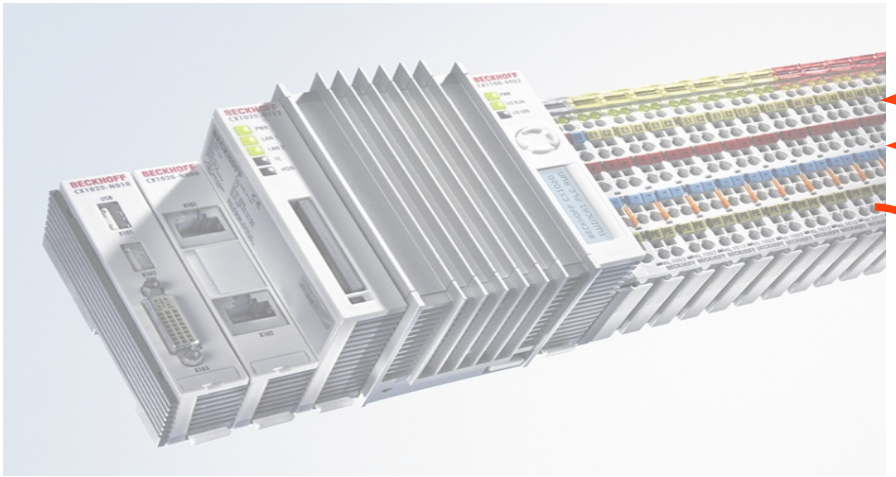
```
CPU  PID    MSW        CSW        PF    STAT       %CPU  NAME
  0  0      0          31831032   0     00500088   42.8  ROOT/0
  0  757    2          4          0     00300182    0.0  bench_main
  0  763    19         23         0     00300182    0.0  bench_scope
  0  764    1          2          0     00300182    0.0  bench_Event
  0  773    2          31831009   0     00300180   56.9  bench
  0  762    1          1          0     00300380    0.0  bench_viewer
  0  0      0          92096      0     00000000    0.1  IRQ42: pio0
  0  0      0          699086     0     00000000    0.0  IRQ52: [timer]
```

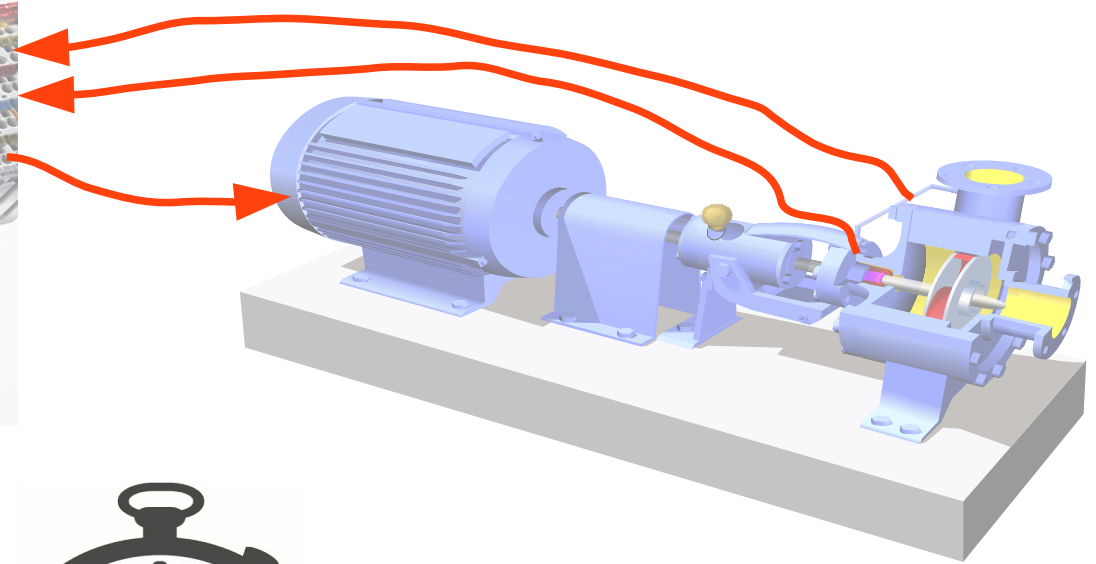
- Using `PTHREAD_WARNSW`: sends a `SIGXCPU` signal
- http://xenomai.org/2014/08/porting-a-linux-application-to-xenomai-dual-kernel

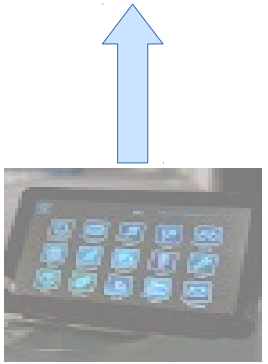
CPU affinity and isolcpus trades off performance vs jitter



Xenomai separates real-time from Linux scheduler







50µs

Implement RT drivers with RTDM framework

- Real-time drivers should work in Xenomai domain not Linux domain
- No access to Linux functions that schedule

  → mode switch
- Real-Time Device Model provides
  - thread (task) operations
  - synchronisation
  - interrupts
  - ...

Userspace API for RTDM drivers

- rt_dev_open(), rt_dev_write(), …
- Avoid mode switch on device open

  read() and write() on RTDM device
  are handled by POSIX skin

- Makes code non-portable
  but stub is anyway needed

Various problems encountered with Xenomai

- x86 SMI workaround ⇒ overheating
- Mix of different skins (RTDM, native, posix)
  ⇒ harder to maintain
- Extra code for simulation on non-RT PC
- No valgrind

PREEMPT_RT

Real-Time in Linux

GPS receiver

septentrio

Measurements sampled @3kHz

GPS receiver with connectivity

septentrio

Wi-Fi ONBOARD

Measurements sampled @3kHz

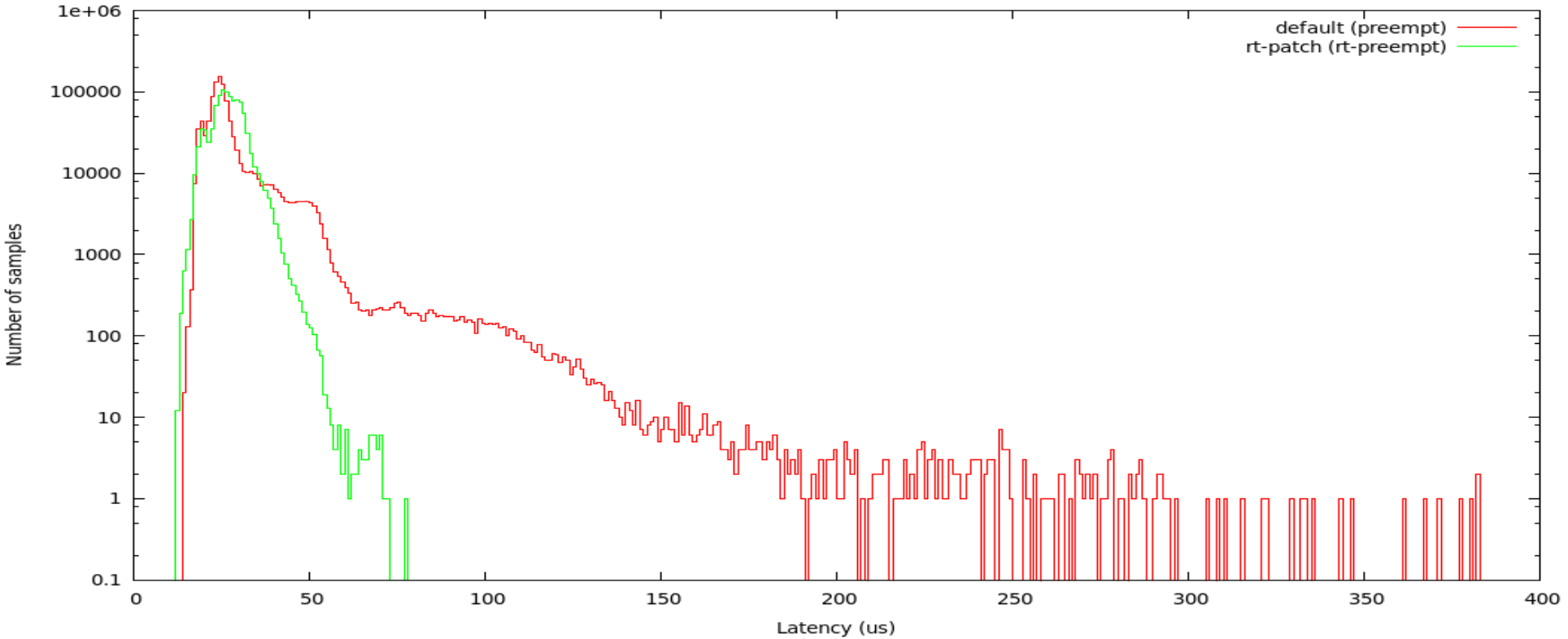
PREEMPT_RT is close to Linux

- http://git.kernel.org/cgit/linux/kernel/git/rt
- Pure kernel implementation, no API/ABI change
- On its way upstream

  slowed down since 3.2 but should be picking up again
- Continuous testing in the OSADL QA farm

  http://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html
- Main difference: every interrupt is a thread

PREEMPT_RT removes almost all disabling of interrupts



Source: http://www.emlid.com/raspberry-pi-real-time-kernel/

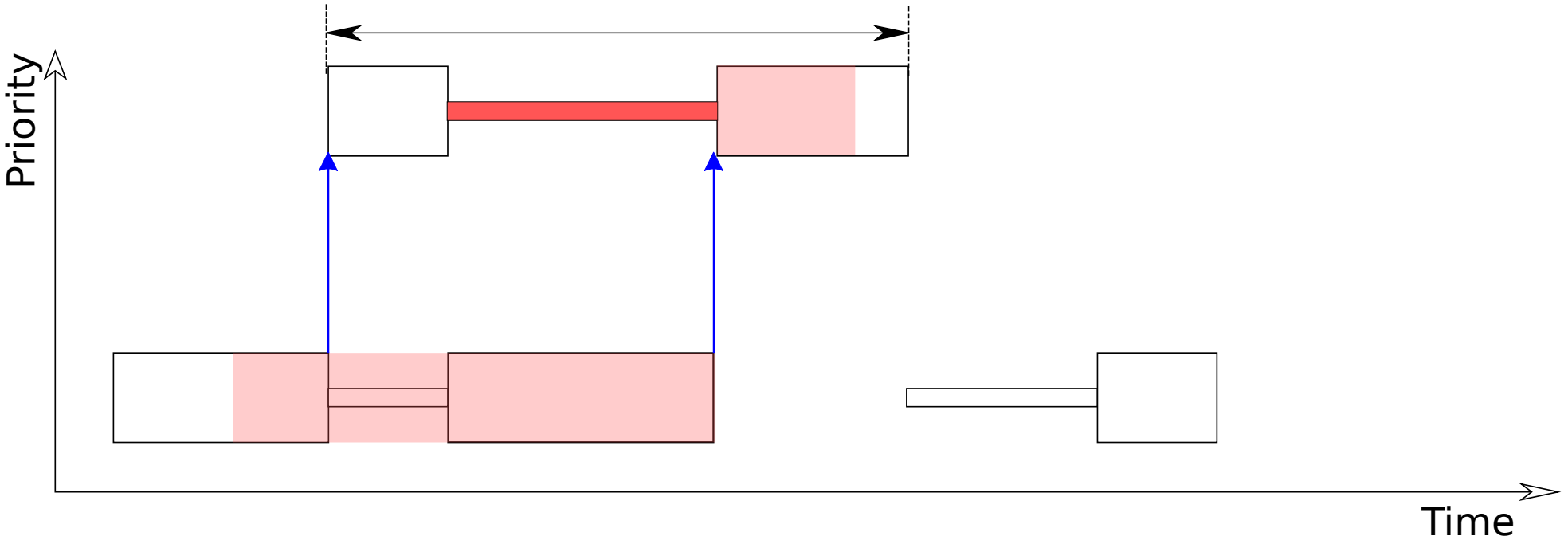
Debugging real-time issues is more difficult with PREEMPT_RT

- No atomic process list

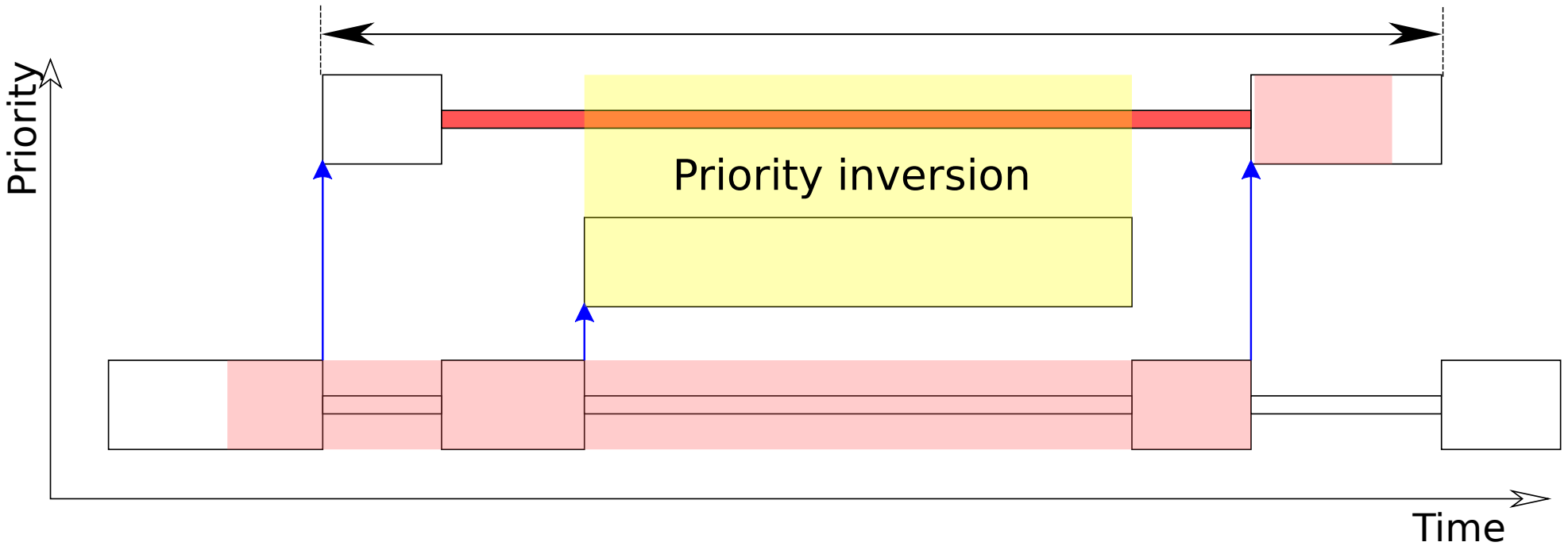
```
top - 08:40:38 up 40 min,  1 user,  load average: 1.01, 1.02, 0.93
Cpu(s): 10.4%us,  8.6%sy,  0.0%ni, 81.0%id,  0.0%wa,
  PID USER      PR S %CPU %MEM    TIME+  COMMAND
  694 root     -61 S 26.4 29.4   9:41.71 pollingThread
  593 root     -51 S  5.3  0.0   2:09.29 irq/288-pm_wkup
  691 root      -7 S  5.3 29.4   2:04.05 thrDNPR
  737 root      20 R  5.3  1.7   0:00.07 top
```

- No “mode switches” to detect priority inversion

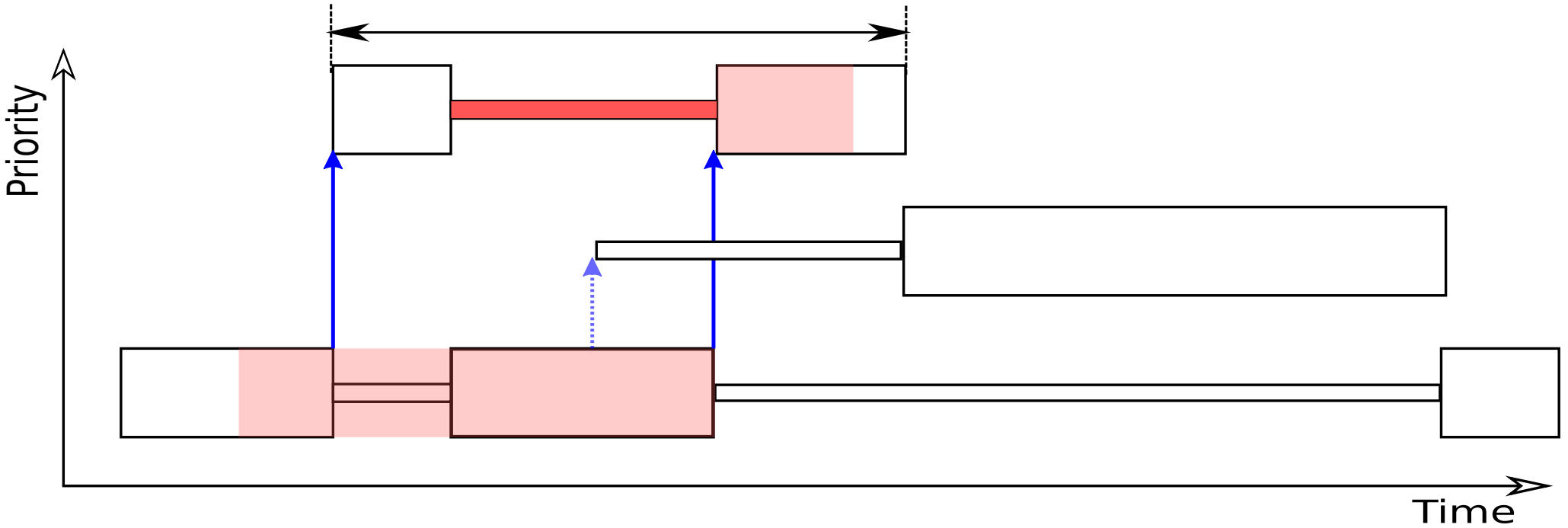
Priority inversion problem



Priority inversion problem



Priority inheritance solves inversion but is expensive



Normal mutex (95% uncontended): avg 600ns

Priority Inheritance mutex: avg 16µs

Priority inversion issues encountered

- interrupts_disable from RTEMS code had to be replaced with PI-mutex

- open() and socket() take lock on the file descriptor table

  ⇒ open all files at initialisation time or in other thread

Priority inversion issues encountered

- Memory manager lock
  - mlockall(MCL_CURRENT|MCL_FUTURE)
  - pre-fault stacks
  - pre-fault malloc()
- Monitor page faults

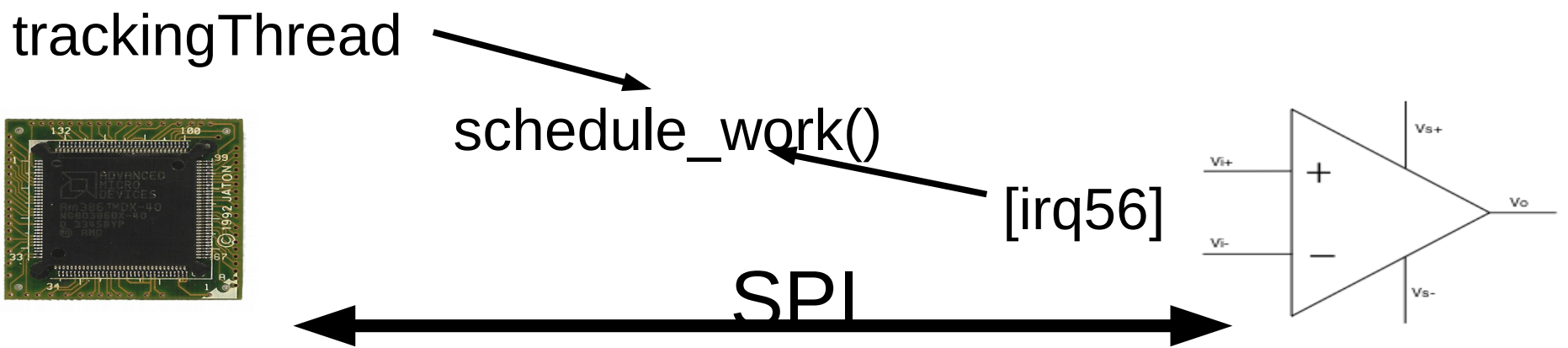
Priority inversion issues encountered

- fork() creates COW references of **all** pages
  Difficult to discover
  - Use vfork()
    → fork() is implicit in system()
  - Move hard real-time code to separate process
    → a lot of work
  - Create system()-wrapper that calls separate process over D-Bus

    Note: dbus-daemon does *not* listen on TCP port :-)

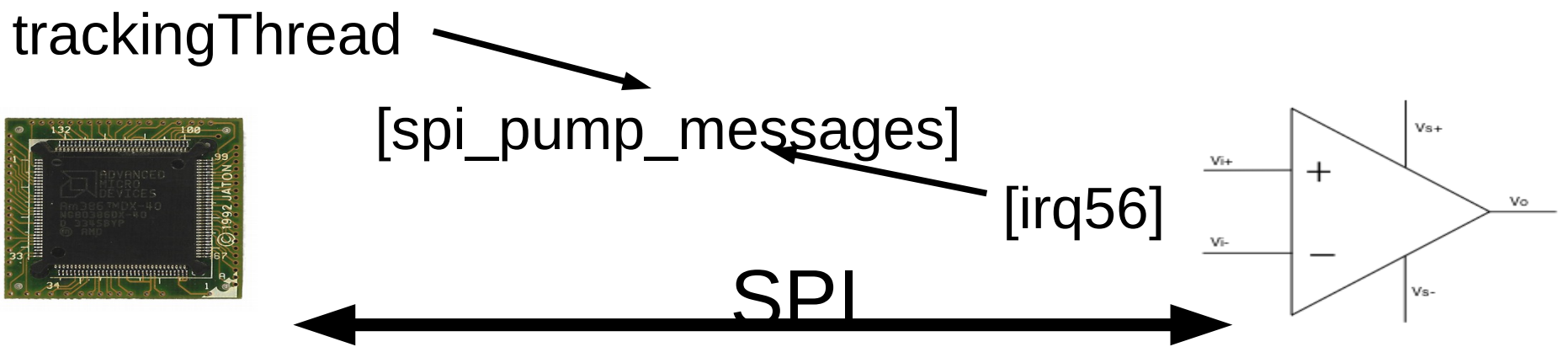
Real-time drivers

- All drivers are “real-time”
- Play with IRQ and kthread priority
- kworker is not associated with specific driver
  ⇒ convert to kthread



Real-time drivers

- All drivers are “real-time”
- Play with IRQ and kthread priority
- kworker is not associated with specific driver
  ⇒ convert to kthread



Conclusions and future directions

Xenomai is converging with PREEMPT_RT

- The Linux kernel running under Xenomai can be PREEMPT_RT
  - Linux ROOT gets currently running thread's priority
- Xenomai 3.x offers dual-kernel and native option
  - Use same “Alchemy” API on Xenomai and PREEMPT_RT
- Xenomai latency is still significantly better
- RTnet only supports Xenomai (RTDM)

Xenomai or PREEMPT_RT?

Xenomai

- Tight latency requirement
- Real-time networking requirement (RTnet, Ethercat)
- Migrating existing RTOS application

PREEMPT_RT

- Use existing drivers from RT threads
- Run same application on different platforms
- Migrating existing POSIX application